**ВВЕДЕНИЕ**

На сегодняшний день применение онлайн-сервисов приобрело весьма большое значение для многих пользователей и компаний, которые для упрощения своей работы применяют компьютерные технологии.

Информационные технологии проникли во все сферы жизни. Они не обошли стороной и автомобили. Производители авто начали внедрять ИТ-технологии в свои новые модели. Автомобили оснащаются мультимедийными системами с выходом в Интернет, адаптивными системами круиз-контроля, камерами кругового обзора, системами автоматической парковки, контроля слепых зон, системами активного ограничения скорости, динамической стабилизации рулевого управления, удержания полосы движения, распознавания дорожных знаков и другими. Кроме того, многие водители в поездках пользуются навигаторами, регистраторами, мобильными телефонами, облегчающими жизнь водителей.

В настоящее время, несмотря на повышение компьютеризации общества, до сих пор нет средств и сервисов, позволяющих предсказать погоду, основываясь на местоположении пользователя, и ответить на злободневный вопрос всех автолюбителей: «Мыть машину или не мыть? ». Одной из основных задач можно рассматривать проблему предсказания погоды, оперативную обработку полученных данных и выдаче результатов пользователю на их основе.

О **своевременности** и **актуальности** рассматриваемой проблемы говорит тот факт, что практически каждый водитель хотя бы раз, наверное, попадал в ситуацию, когда только помыл машину и пошел дождь. Приложение «GoodAdvice» является надежным персональным синоптиком. Его преимущество в том, что при помощи системы геолокации, оно определяет местоположение и дает точный прогноз погоды на несколько дней. На основе этого прогноза приложение само подсказывает: мыть машину или отложить это действие на сколько дней.

**Целью** данного курсового проекта является создание приложения, которое может получать данные от метеослужб, обрабатывать, анализировать их и выдавать заключение на их основе, помогая пользователю (водителю) сделать правильный выбор и определить день, который лучше всего подойдет для того, чтобы помыть автомобиль.

**Основные** **задачи**:

* Взаимодействие с API метеослужб. Предсказание погоды на ближайшие дни.
* Предоставление информации пользователю посредством удобного интерфейса.

В данной курсовой работе будут использована система Windows Presentation Foundation (WPF) — система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, сторонние библиотеки (Awesomium, HamburgerMenu, HtmlAgilityPack) и сторонние API.

**1 Анализ проблемы**

Для того чтобы онлайн-сервис пользовался успехом, необходимо, чтобы он правильно и достоверно работал. Поэтому, конечно, необходимо точно определить местоположение пользователя, а уже потом осуществить корректное прогнозирование и дать совет, стоит ли мыть автомобиль.

Сразу после запуска любого приложения появляется главное окно, которое содержит приветствие и приглашение пользователя к дальнейшим действиям.

Логика приложения должна быть грамотно продумана, чтобы любые необдуманные действия пользователя не привели к краху сервиса. Поэтому в программе предусмотрены различные, так называемые, проверки от ошибочных действий пользователя.

Местоположение пользователя определятся в 3 этапа. Сначала оно определяется через встроенный GPS модуль. В случае ошибки, приложение пытается определить геолокацию по IP адресу. Если и через IP адрес вычислить не удалось, то система просит пользователя ввести местоположение вручную.

В качестве сторонних API, сервис использует следующие: Yandex Geocoder API, Yandex JavaScript API, Yandex Translate API.

В качестве сторонних библиотек и пакетов: HtmlAgilityPack, HamburgerMenu.

* 1. **Анализ функциональных требований**

Для реализации поставленной задачи необходимо рассмотреть функции, которые имеет, так называемый, «сервер» приложения (рисунок 1).

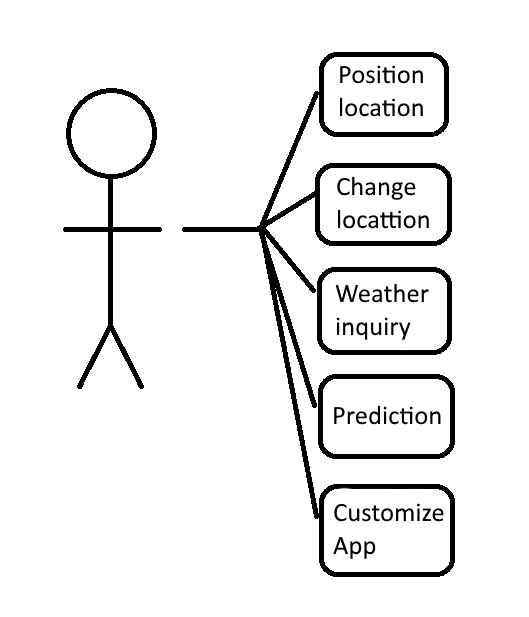


Рисунок 1 – Действия сервера

Для сервера доступны следующие функции:

1. Position location – определять текущее местоположение пользователя
2. Change location – изменять текущее местоположение, так как клиент может перемещаться.
3. Wheather inquiry – запрашивать и анализировать погодные данные.
4. Prediction – по собранным метеоданным осуществлять прогнозирование и выдавать совет.
5. Customize App – настраивать приложение (например, сохранять настройки местоположения после завершения работы программы).
   1. **Используемые технологи**

**API** (интерфейс прикладного программирования) (англ. application programming interface, API) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах. Используется программистами при написании всевозможных приложений.

**Yandex JavaScript API** – набор компонентов для размещения интерактивных Яндекс.Карт на страницах сайта или в веб-приложении. С помощью API можно отображать карты с различными объектами, искать адреса, прокладывать маршруты, строить свои схемы и многое другое. Интерфейс требует поддержки JavaScript.

**Yandex Geocoder API –** компонент API Яндекс.Карт, с помощью которого определяются координаты объекта на карте по его адресу, и наоборот. Например, по запросу [Москва, ул. Малая Грузинская, д. 27/13] геокодер вернёт географические координаты этого дома: 37.571309  ° восточной долготы, 55.767190  ° северной широты. А если в запросе указать географические координаты нужной точки [27.525773, 53.89079], то геокодер вернет её адрес: Минск, проспект Дзержинского, 5.

В бесплатной версии API Карт можно делать не более 25 000 запросов к геокодеру в сутки.

**Yandex Translate API –** c помощью API можно получить доступ к онлайн-сервису машинного перевода Яндекса. Он поддерживает более 70 языков и умеет переводить отдельные слова и целые тексты. Этот API позволяет встроить Яндекс.Переводчик в мобильное приложение или веб-сервис для конечных пользователей. Или же переводить большие объёмы текста — например, техническую документацию.

**HtmlAgilityPack** – один из самых (если не самый) известный парсер HTML в мире .NET. Быстрая, относительно удобная библиотека для работы с HTML (если XPath запросы будут несложными). Парсер будет удобным если задача типична и хорошо описывается XPath выражением, к примеру, чтобы получить все ссылки со страницы, нам понадобится совсем немного кода:

Библиотека **Hamburger Menu** помогает просто реализовать динамическое меню приложения любой сложности. Своё название она получила благодаря кнопке, которая это меню вызывает. В WPF-приложениях нет стандартных средств, с помощью которых можно создать это меню, в отличие от UWP-приложений.

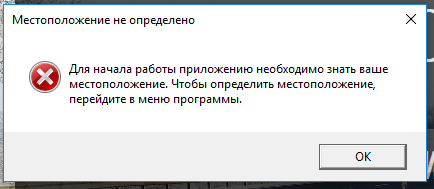
**Awesomium** — это библиотека, для интеграции браузера на базе Chromium в своё приложение. Достоинство Awesomium состоит в том, что его можно интегрировать в приложение практически любого типа, он обладает широким набором возможностей для разработчика.

**2 Реализация**

Начнем с самого начала, а именно с того, как только пользователь запустил наше приложение. Выше уже говорилось о защите от ошибочных действий пользователя. Теперь разберем это более подробно. Если пользователь захочет получить совет, не указав программе своё местоположение, программа выдаст ошибку. Ниже представлен алгоритм реализации вызова данного исключения:

|  |
| --- |
| if (((App)Application.Current).cityName == "Не определено")  {  MessageBox.Show("", MessageBoxButton.OK, MessageBoxImage.Error);  MessageBox.Show("", MessageBoxButton.OK, MessageBoxImage.Information);  textBlock1.Text = "А вот и меню\n↓↓↓";  pLink.IsOpen = true; // запускаем всплывающее окно  } |

Сгенерированное исключение показано на рисунке 2:



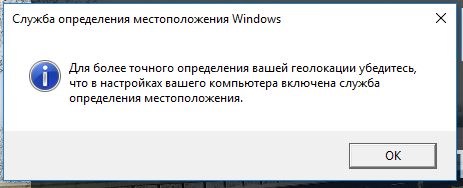


Рисунок 2 – Защита от ошибочных действий пользователя

Теперь пользователю становится понятно, что нужно указать приложению своё местоположение. Ниже представлен код определения геолокации юзера:

|  |
| --- |
| private string SearchCurrentLocation()  {  List<double> coordinates = SearchCurrentLocationViaGPS();  string cityName = "";  if (coordinates[0] != -999)  {  try  {  string longitude = coordinates[0].ToString().Replace(",", ".");  string latitude = coordinates[1].ToString().Replace(",", ".");  cityName = GeocodeCoordinatesInTheNameOfTheCity("https://" + $"geocode-maps.yandex.ru/1.x/?geocode={longitude},{latitude}&kind=locality&results=1.");  }  catch  {  coordinates.Clear();  coordinates.Add(-999);  }  }  //Если не удалось определить местоположение по GPS, пробуем вычислить по IP  if (coordinates[0] == -999)  {  try  {  coordinates = SearchCurrentLocationViaIP();  string longitude = coordinates[0].ToString().Replace(",", ".");  string latitude = coordinates[1].ToString().Replace(",", ".");  cityName = GeocodeCoordinatesInTheNameOfTheCity("https://" + $"geocode-maps.yandex.ru/1.x/?geocode={longitude},{latitude}&kind=locality&results=1.");  }  catch  {  cityName = "none";  }  }  return cityName;  }  //Поиск местоположения через GPS  private List<double> SearchCurrentLocationViaGPS()  {  var watcher = new GeoCoordinateWatcher();  watcher.TryStart(false, TimeSpan.FromMilliseconds(1000));  var coord = watcher.Position.Location;  List<double> answer = new List<double>();  Stopwatch stopWatch = new Stopwatch();  stopWatch.Start();  while (coord.IsUnknown == true)  {  coord = watcher.Position.Location;  TimeSpan ts = stopWatch.Elapsed;  int temp = (int)ts.TotalMilliseconds;  if (temp >= 6000)  {  answer.Add(-999);  return answer;  }  }  stopWatch.Stop();  answer.Add(coord.Longitude);  answer.Add(coord.Latitude);  return answer;  }  private List<double> SearchCurrentLocationViaIP()  {  List<double> answer = new List<double>();  var locationResponse = new WebClient().DownloadString("https://freegeoip.net/xml/");  var responseXml = XDocument.Parse(locationResponse).Element("Response");  string longitude = responseXml.Element("Longitude").Value;  string latitude = responseXml.Element("Latitude").Value;  longitude = longitude.Replace(".", ",");  latitude = latitude.Replace(".", ",");  //Преобразуем кординаты в тип double и возвращаем  answer.Add(double.Parse(longitude));  answer.Add(double.Parse(latitude));  return answer;  } |

Можно понять, что в данном коде, функции которые определяют местоположение либо через GPS модуль, либо через IP адрес, возвращают нам географические координаты [1]. Для запроса погодных данных нам необходимо конкретное название населенного пункта. Поэтому, для преобразования географических координат в название города, используем стороннюю API: Yandex Geocoder API.

|  |
| --- |
| //Геоодирование координат в название города  private string GeocodeCoordinatesInTheNameOfTheCity(string reguest)  {  XmlDocument xd = new XmlDocument();  //Отправляем запрос Яндекс.Геокодеру и загружаем ответ в виде xml  xd.Load(reguest);  //Начинаем проходить по тегам xml документа и считываем информацию  XmlNode ymaps = xd.DocumentElement;  //Тэг "Country"  XmlNodeList GeoObjectTempCountry = xd.GetElementsByTagName("Country");  string addressLine = "",  countryNameCode = "",  countryName = "",  administrativeAreaName = "",  localityName = "";  foreach (XmlNode countryNode in GeoObjectTempCountry)  {  foreach (XmlNode childCountry in countryNode.ChildNodes)  {  switch (childCountry.Name)  {  case "AddressLine":  addressLine = childCountry.FirstChild.InnerText;  break;  case "CountryNameCode":  countryNameCode = childCountry.FirstChild.InnerText;  break;  case "CountryName":  countryName = childCountry.FirstChild.InnerText;  break;  }  }  //Ребенок "AdministrativeArea"  XmlNodeList GeoObjectTempAdministrativeArea = xd.GetElementsByTagName("AdministrativeArea");  foreach (XmlNode areaNode in GeoObjectTempAdministrativeArea)  {  foreach (XmlNode childAreaName in areaNode.ChildNodes)  {  switch (childAreaName.Name)  {  case "AdministrativeAreaName":  administrativeAreaName = childAreaName.FirstChild.InnerText;  break;  }  }  }  //Ребенок "Locality"  XmlNodeList GeoObjectTempLocality = xd.GetElementsByTagName("Locality");  foreach (XmlNode localityNode in GeoObjectTempLocality)  {  foreach (XmlNode childLocalituName in localityNode.ChildNodes)  {  switch (childLocalituName.Name)  {  case "LocalityName":  localityName = childLocalituName.FirstChild.InnerText;  break;  }  }  }  }  return localityName;  } |

В результате мы получаем конкретное название населенного пункта. В случае если произошел какой-то сбой, в качестве название населенного пункта, приложение вернет строку «**none**» для дальнейшей удобной обработки.

После определения геолокации начинаются многочисленные проверки. В том числе проверка на интернет соединения. Если интернет соединения не активно, то пользователь не сможет ни определить местоположение, ни изменить его.

При этом логика программы построена таким образом, что если местоположение уже определено, то повторное его определение вызовет ошибку (рисунок 3).

|  |
| --- |
| // случай когда местоположение уже определяли  else if (hasLocation == true && ((App)Application.Current).cityName != "change")  {  MainWindow wnd = (MainWindow)App.Current.MainWindow;  wnd.hamMenu.Visibility = System.Windows.Visibility.Visible;  PB.Visibility = System.Windows.Visibility.Collapsed;  viewBox1.Visibility = System.Windows.Visibility.Collapsed;  picture.Visibility = System.Windows.Visibility.Collapsed;  MessageBox.Show("Ваше местоположение уже определено!", "Ошибка #001", MessageBoxButton.OK, MessageBoxImage.Error);  wnd.item1.IsSelected = true; // переходим на главную страницу  } |

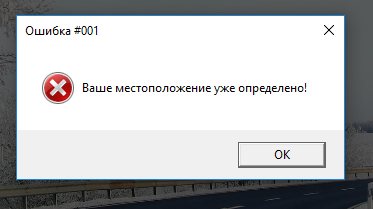


Рисунок 3 – Защита от ошибочных действий пользователя

Может сложиться такая ситуация, что на устройстве отсутствует GPS модуль [2]. В таком случае, местоположение будет определяться по IP. Так как провайдер может находиться в абсолютно другом городе, пользователь может задать местоположение вручную (рисунок 4):

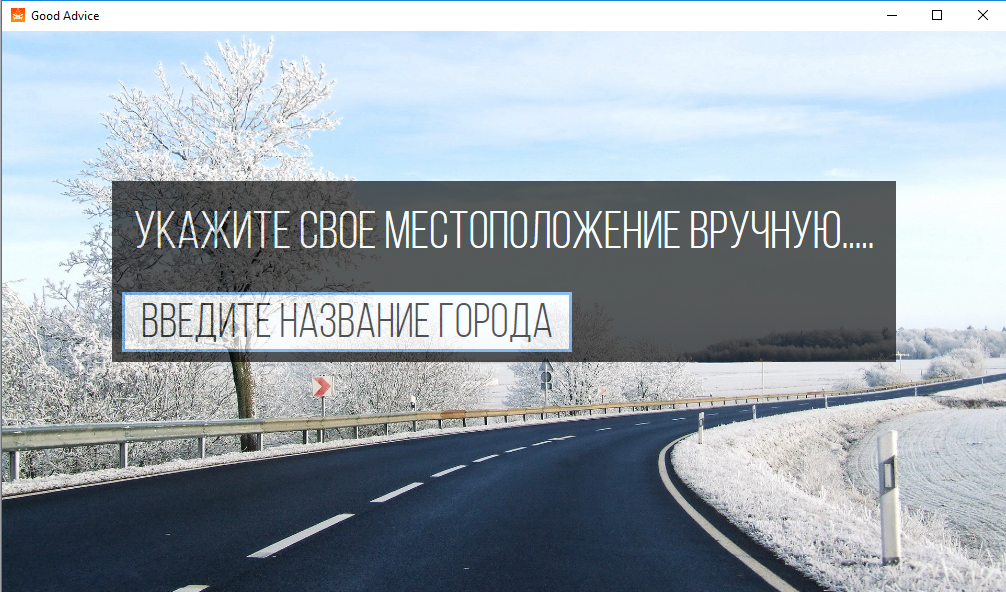


Рисунок 4 – ручное задание местоположения

Сразу может возникнуть вопрос, что пользователь может ввести название города с ошибкой, поэтому необходимо предусмотреть проверку на правильность ввода. Для этого подключим к нашему приложению базу из 1223 российских городов. И при любом изменении нашего textbox’а будем проверять, содержится ли значение textbox’аd нашей базе. Если да, то немедленно уведомим об этом пользователя.

Ниже представлен класс, реализующий данную проверку:

|  |
| --- |
| //список городов  List<String> citiesName = new List<string>();  //считываем список городов в конструкторе класса из файла  public CheckCityName()  {  using (System.IO.StreamReader file = new System.IO.StreamReader("checkCityName.ini"))  {  string temp = file.ReadToEnd();  file.Close();  String[] subtemp = temp.Split('\n');  for (int i = 0; i < subtemp.Length; ++i)  {  if (subtemp[i].Contains("\r"))  {  subtemp[i] = subtemp[i].Remove(subtemp[i].Length - 1, 1);  citiesName.Add(subtemp[i].ToUpper());  }  else  {  citiesName.Add(subtemp[i].ToUpper());  }  }  }  }  //проверяем наличие города с таким именем  public bool CheckCitiesName(string cityName)  {  if (citiesName.Contains(cityName))  return true;  else  return false;  } |

В данном приложении, есть раздел меню «Настройки», где можно сохранить текущее местоположение после завершения программы. Текущее местоположение сохраняется в текстовый файл и считывается при начальной загрузки формы. Для считывания и записи информации в файл, реализуем interface ILOG и опишем класс TextFileLogger:

|  |
| --- |
| interface ILog  {  void Log(string Message);  string Read();  }  public class TextFileLogger : ILog  {  public void Log(string Message)  {  using (System.IO.StreamWriter file = new System.IO.StreamWriter(\_name, false))  {  file.WriteLine(Message);  file.Close();  }  }  public string Read()  {  string temp = System.String.Empty;  using (System.IO.StreamReader file = new System.IO.StreamReader(\_name))  {  temp = file.ReadLine();  }  return temp;  }  private string \_name;  public TextFileLogger(string name)  {  this.\_name = name;  }  } |

После определения местоположения, программа выводит текущий прогноз погоды с помощью RSS ленты сторонней API, а именно GISMETEO API. Основная проблема состоит в том, что при отправлении запроса к данной API, необходимо вместо название города, указать его ZIP код [3]. Для этого опишем ещё один класс, который будет именно этим и заниматься.

|  |
| --- |
| public class ZIPCodeBYCityName  {  //словарь содержит название города и его zip code для запроса прогноза погоды  List<KeyValuePair<string, string>> dictionary = new List<KeyValuePair<string, string>>();  //метод заполнение словаря  public void FillDictionary()  {  string temp = System.String.Empty;  //считываем инфомацию из файла  using (System.IO.StreamReader file = new System.IO.StreamReader("initialZIPCodeAndCityName.ini"))  {  temp = file.ReadToEnd();  file.Close();  //сплитим по переносу строки  String[] subtemp = temp.Split('\n');  for (int i = 0; i < subtemp.Length; ++i)  {  //вырезаем не нужные символы и добавлям элементы  if (subtemp[i].Contains("\r"))  {  subtemp[i] = subtemp[i].Remove(subtemp[i].Length - 1, 1);  dictionary.Add(new KeyValuePair<string, string>(subtemp[i].Split(' ')[1].ToUpper(), subtemp[i].Split(' ')[0]));  }  else  {  dictionary.Add(new KeyValuePair<string, string>(subtemp[i].Split(' ')[1].ToUpper(), subtemp[i].Split(' ')[0]));  }  }  }  }  //метод ищет zip code переданного ему в качестве парамметра названия города  public int SearchZIPCodeByCityName(string cityName)  {  KeyValuePair<string, string> temp = dictionary.First(v => v.Key.Equals(cityName));  return Int32.Parse(temp.Value);  }  } |

Логикой программы, предусмотрено, что если для текущего города не удается получить ZIP код, программа скажет, что для данного региона прогноз погоды не доступен.

Передача данных в приложении между страницами осуществляется посредством главного файла App.xaml.cs. На странице с ближайшими автомойками, добавлен компонент формы Awesomium, на котором, с помощью JavaScript API показываются ближайшие к текущему местоположению пользователя автомойки.

Теперь поговорим об основной части нашего приложения – это прогнозирование полученных метеоданных и выдача совета. В данном случае запрос погоды осуществляется через обычный WEB запрос к сервису Яндекс.Погода. В результате мы получаем HTML страничку, которую парсим и достаем необходимые нам элементы с помощью библиотеки HtmlAgilityPack [4]. Для хранения необходимых метеоданных опишем класс с нужными полями.

|  |
| --- |
| public class Wheather  {  public Wheather()  {  tempDay = 0;  cloudiness = "";  }  public int tempDay;  public string cloudiness;  }  public Prediction(string \_cityName)  {  temp = GetTranslit(\_cityName);  }  public string temp;  public string url;  List<Wheather> forecast = new List<Wheather>();  List<string> answer = new List<string>();  int ret = 0;  public KeyValuePair<string, int> Advice()  {  url = @"https://yandex.ru/pogoda/" + $"{temp}";  resizeForecast();  resizeAnswer();  SetData(url, "forecast-brief\_\_item-comment");  SetData(url, "forecast-brief\_\_item-temp-day");  return new KeyValuePair<string, int>(answer[PredictionAnswer()], ret);  }  private void resizeForecast()  {  for (int i = 0; i < 5; ++i)  {  Wheather wh = new Wheather();  wh.tempDay = 0;  wh.cloudiness = "";  forecast.Add(wh);  }  }  private void resizeAnswer()  {  answer.Add("Пока нет ни одного подходящего дня для мойки вашего автомобиля. Придется немного подождать.");  answer.Add("Самое время быстрее ехать на мойку и мыть свою ласточку.");  answer.Add("Можете помыть машину сегодня и как минимум ");  answer.Add("Можете помыть машину завтра и как минимум ");  answer.Add("Можете помыть машину послезавтра и как минимум ");  }  private void SetData(string url, string className)  {  HtmlDocument HD = new HtmlDocument();  var web = new HtmlWeb  {  AutoDetectEncoding = false,  OverrideEncoding = Encoding.UTF8,  };  HD = web.Load(url);  //выбирае деревья из класса написанного в textBox и элемента написанного  HtmlNodeCollection NoAltElements = HD.DocumentNode.SelectNodes($"//div[@class='{className}']");  int i = 0;  if (NoAltElements != null)  {  foreach (HtmlNode hn in NoAltElements)  {  string outputText = hn.InnerText.Trim();  if (className == "forecast-brief\_\_item-comment")  {  forecast[i].cloudiness = outputText;  i++;  if (i == 5)  break;  }  else  {  if (i == 0)  {  outputText = outputText.Remove(2, outputText.Length - 2);  forecast[i].tempDay = Int32.Parse(outputText[1].ToString());  if (outputText[0] == '−')  forecast[i].tempDay = -forecast[i].tempDay;  }  else  {  if (outputText.Length > 1)  {  forecast[i].tempDay = Int32.Parse(outputText[1].ToString());  if (outputText[0] == '−')  forecast[i].tempDay = -forecast[i].tempDay;  }  else  forecast[i].tempDay = Int32.Parse(outputText);  }  i++;  if (i == 5)  break;  }  }  }  } |

При отправке WEB запроса к сервису Яндекс.Погода необходимо передавать имя города на транслите, поэтому необходимо перевести текущее местоположение. Это делается с помощью сторонней API: Yandex Translate API:

|  |
| --- |
| public static string GetTranslit(string sourceText)  {  WebRequest request = WebRequest.Create("https://translate.yandex.net/api/v1.5/tr.json/translate?"  + "key=trnsl.1.1.20161223T182008Z.9d1ae536335cebe1.7d22d7829ce46f43568f157baf83b7c39b470297"  + "&text=" + sourceText  + "&lang=" + "ru-en");  WebResponse response = request.GetResponse();  using (StreamReader stream = new StreamReader(response.GetResponseStream()))  {  string line;  if ((line = stream.ReadLine()) != null)  {  sourceText = line.Substring(line.IndexOf(":[\"") + 3);  sourceText = sourceText.Remove(sourceText.Length - 3);  }  }  return sourceText;  } |

После того, как мы получили метеоданные и преобразовали их в нормальный для обработки вид, переходим к прогнозированию [5].

Алгоритм:

1. Имеем метеоданные на 5 дней.
2. Начинаем обходить их с конца, при этом анализируем каждый день по очереди. Если делаем вывод, что день пригоден для помывки автомобиля, то увеличиваем счетчик. Переходим к следующему дню.
3. Если день не подходит для мытья авто, то обнуляем счетчик, за исключением того, что счетчик уже >= 3. В этом случае уже можно выдать совет.

|  |
| --- |
| private int PredictionAnswer()  {  int ans = 0;  int pos = 0;  for (int i = 4; i >= 0; i--)  {  if (i > 0)  {  if (CheckCloudiness(forecast[i].cloudiness) && CheckTempDay(forecast[i].tempDay, forecast[i - 1].tempDay))  ret++;  else  {  if (ret >= 3)  {  pos = i + 2;  break;  }  ret = 0;  }    }  if (i == 0)  {  if (CheckCloudiness(forecast[i].cloudiness))  {  if (ret >= 2)  {  ret++;  pos = 1;  }  }  else  {  if (ret >= 3)  {  pos = i + 2;  break;  }  }  }  }  if (pos == 1 && ret == 5)  ans = 1;  if (ret < 3)  ans = 0;  if (pos == 1 && ret != 5)  ans = 2;  if (pos == 2)  ans = 3;  if (pos == 3)  ans = 4;  return ans;  }  private bool CheckCloudiness(string temp)  {  bool hasCheckCloudiness = true;  if (temp == "небольшой дождь")  hasCheckCloudiness = false;  if (temp == "дождь")  hasCheckCloudiness = false;  if (temp == "сильный дождь")  hasCheckCloudiness = false;  if (temp == "сильный дождь, гроза")  hasCheckCloudiness = false;  if (temp == "дождь со снегом")  hasCheckCloudiness = false;  if (temp == "небольшой снег")  hasCheckCloudiness = false;  if (temp == "снег")  hasCheckCloudiness = false;  if (temp == "снегопад")  hasCheckCloudiness = false;  return hasCheckCloudiness;  }  private bool CheckTempDay(int temp1, int temp2)  {  bool hasCheckTempDay = true;  if (temp1 < 0 && temp2 > 0)  hasCheckTempDay = false;  return hasCheckTempDay;  } |

Ниже представлен рисунок с всевозможными результатами работы программы.

На рисунке 5 показан результат работы программы, в случае, когда машину мыть не стоит:

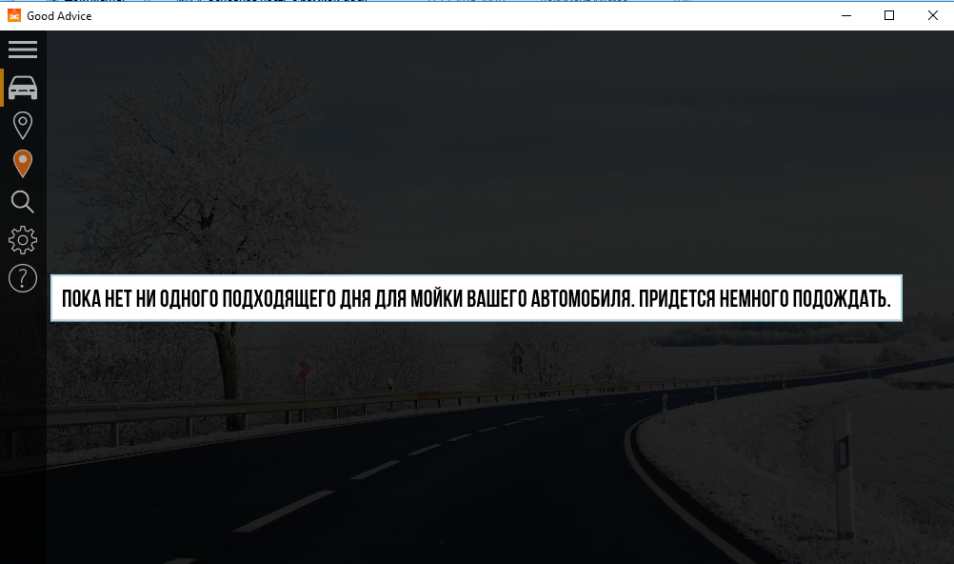


Рисунок 5 – результаты работы программы

На рисунке 6 показан результат работы программы, в случае, когда машину можно мыть:

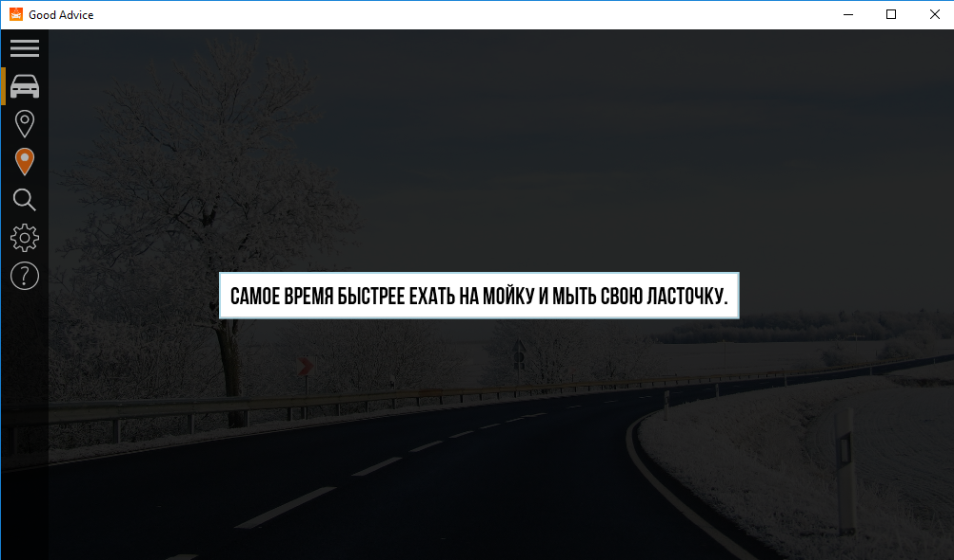


Рисунок 6 – результаты работы программы

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы была разработана грамотная логика для онлайн-сервиса на язык **С#**. Приложение позволяет определить местоположение пользователя либо через GPS модуль, либо через IP адрес. Можно задать местоположение вручную и изменять его, просмотреть ближайшие автомойки. Сервис умеет запрашивать, анализировать и выводить в удобном для пользователя виде погоду. Основная функция приложения – это прогнозирование полученных метеоданных и выдача советов на их основе о том, стоит ли мыть машину.

Были изучены сторонние библиотеки (Awesomium, Hamburger Menu, HtmlAgilityPack), новые технологии, в том числе работа с **API** (Yandex API, OpenWeatherMap, GisMeteo), новые возможности языка высокого уровня C#. Также были изучены более подробно формы WPF и специфичные библиотеки, дополняющие стандартные возможности языка.

Все поставленные задачи были выполнены.

* Взаимодействие с API метеослужб. Предсказание погоды на ближайшие дни. (Gismeteo API, Яндекс.Погода API). Приложение запрашивает метеоданные и выводит их в удобном для пользователя виде.
* Предоставление информации пользователю посредством удобного интерфейса. Предоставление совета, посредством прогнозирования. Обеспечение быстродействия программы за счет оптимального алгоритма и грамотной продуманной логики приложения.

В ходе выполнения данной работы на основе различных источников были повышены знания и навыки в области ООП.

В процессе тестирования приложения ошибок не обнаружено.

**Список использованных источников**

1. Шилдт Г. Полный справочник по С# 4.0 / Г. Шилдт. – Лондон: «Osborne», 2010. –112 с.
2. М. Мак-Дональд WPF. Windows Presentation Foundation в .NET 3.5 с примерами на C# 2008 для профессионалов/ Вильямс. – 928 c.
3. Metanit: [Электронный ресурс]. URL: <http://metanit.com>
4. Metanit: [Электронный ресурс]. URL: <http://www.awesomium.com>
5. Yandex API: [Электронный ресурс]. URL: https://tech.yandex.ru/